In [2147]:

```
using Plots
using CSV
using DataFrames
using Interpolations
```

In [2148]:

```
# Import battery performance data (taken from graphs and generated by Engauge)
csv_diamond_05C = CSV.File("Foxtech Diamond/Diamond cell test, 0.5C.csv")
df_diamond_05C = DataFrame(csv_diamond_05C)
itp_diamond_05C = LinearInterpolation(df_diamond_05C.Ahr, df_diamond_05C.V, extrapolation_b

csv_diamond_3C = CSV.File("Foxtech Diamond/Diamond cell test, 3C.csv")
df_diamond_3C = DataFrame(csv_diamond_3C)
itp_diamond_3C = LinearInterpolation(df_diamond_3C.Ahr, df_diamond_3C.V, extrapolation_bc=L

csv_diamond_5C = CSV.File("Foxtech Diamond/Diamond cell test, 5C.csv")
df_diamond_5C = DataFrame(csv_diamond_5C)
itp_diamond_5C = LinearInterpolation(df_diamond_5C.Ahr, df_diamond_5C.V, extrapolation_bc=L

## IMPORTANT TO UPDATE WHENEVER THE BATTERY DATASHEETS ARE CHANGED
## The order is important, it must be done in sequential order by growing discharge rate
discharge_curves_diamond = (
  (0.5, itp_diamond_05C),
  (3.0, itp_diamond_3C),
  (5.0, itp_diamond_5C)
);

# Capacity of cells in datasheet, in [Ahr]
ahr_datasheet_diamond_cell = 22

# Individual cell max discharge rate
c_diamond_max = 5.5
```

Out[2148]:

```
5.5
```

In [2149]:

```
# Import battery performance data (taken from graphs and generated by Engauge)
csv_lipo_0_84C = CSV.File("Molicel/P42A/P42A, 0_84C.csv")
df_lipo_0_84C = DataFrame(csv_lipo_0_84C)
itp_lipo_0_84C = LinearInterpolation(df_lipo_0_84C.Ahr, df_lipo_0_84C.V, extrapolation_bc=L

csv_lipo_4_2C = CSV.File("Molicel/P42A/P42A, 4_2C.csv")
df_lipo_4_2C = DataFrame(csv_lipo_4_2C)
itp_lipo_4_2C = LinearInterpolation(df_lipo_4_2C.Ahr, df_lipo_4_2C.V, extrapolation_bc=Line

csv_lipo_10C = CSV.File("Molicel/P42A/P42A, 10C.csv")
df_lipo_10C = DataFrame(csv_lipo_10C)
itp_lipo_10C = LinearInterpolation(df_lipo_10C.Ahr, df_lipo_10C.V, extrapolation_bc=Line())

csv_lipo_20C = CSV.File("Molicel/P42A/P42A, 20C.csv")
df_lipo_20C = DataFrame(csv_lipo_20C)
itp_lipo_20C = LinearInterpolation(df_lipo_20C.Ahr, df_lipo_20C.V, extrapolation_bc=Line())

csv_lipo_30C = CSV.File("Molicel/P42A/P42A, 30C.csv")
df_lipo_30C = DataFrame(csv_lipo_30C)
itp_lipo_30C = LinearInterpolation(df_lipo_30C.Ahr, df_lipo_30C.V, extrapolation_bc=Line())

## IMPORTANT TO UPDATE WHENEVER THE BATTERY DATASHEETS ARE CHANGED
## The order is important, it must be done in sequential order by growing discharge rate
discharge_curves_P42A = (
  (0.84, itp_lipo_0_84C),
  (4.2,  itp_lipo_4_2C),
  (10.0, itp_lipo_10C),
  (20.0, itp_lipo_20C),
  (30.0, itp_lipo_30C)
);

# Capacity of cells in datasheet, in [Ahr]
ahr_datasheet_P42A_cell = 4.2

# Individual cell mass, in [kg]
mass_P42A_cell = 0.070

# Individual cell max continuous discharge rate. This is taken from testing at https://www.
c_P42A_max = 45/ahr_datasheet_P42A_cell

# Individual cell nominal energy capacity, in [kwhr]
kwhr_P42A_cell = .0147
```

Out[2149]:

```
0.0147
```

In [2150]:

```
# Import battery performance data (taken from graphs and generated by Engauge)
csv_lipo_0_84C = CSV.File("Molicel/P26A/P26A,0_52C.csv")
df_lipo_0_84C = DataFrame(csv_lipo_0_84C)
itp_lipo_0_84C = LinearInterpolation(df_lipo_0_84C.Ahr, df_lipo_0_84C.V, extrapolation_bc=L

csv_lipo_10C = CSV.File("Molicel/P26A/P26A,10C.csv")
df_lipo_10C = DataFrame(csv_lipo_10C)
itp_lipo_10C = LinearInterpolation(df_lipo_10C.Ahr, df_lipo_10C.V, extrapolation_bc=Line())

csv_lipo_12_6C = CSV.File("Molicel/P26A/P26A,12_6C.csv")
df_lipo_12_6C = DataFrame(csv_lipo_12_6C)
itp_lipo_12_6C= LinearInterpolation(df_lipo_12_6C.Ahr, df_lipo_12_6C.V, extrapolation_bc=Li

csv_lipo_20C = CSV.File("Molicel/P26A/P26A,20C.csv")
df_lipo_20C = DataFrame(csv_lipo_20C)
itp_lipo_20C = LinearInterpolation(df_lipo_20C.Ahr, df_lipo_20C.V, extrapolation_bc=Line())

## IMPORTANT TO UPDATE WHENEVER THE BATTERY DATASHEETS ARE CHANGED
## The order is important, it must be done in sequential order by growing discharge rate
discharge_curves_P26C=(
  (0.84, itp_lipo_0_84C),
  (4.2,  itp_lipo_4_2C),
  (10.0, itp_lipo_10C),
  (20.0, itp_lipo_20C),
  (30.0, itp_lipo_30C)
);

# Capacity of cells in datasheet, in [Ahr]
ahr_datasheet_P26_cell = 2.6

# Individual cell mass, in [kg]
mass_P26_cell = 0.05

# Individual cell max discharge rate
c_P26_max = 35/ahr_datasheet_P26_cell

# Individual cell nominal energy capacity, in [kwhr]
kwhr_P26_cell = .095
```

Out[2150]:

```
0.095
```

In [2151]:

```
# Import battery performance data (taken from graphs and generated by Engauge)
# A123Systems cells. Probably shouldn't be changed
csv_a123_1C = CSV.File("A123Systems/ANR26650M1B, 1C.csv")
df_a123_1C = DataFrame(csv_a123_1C)
itp_a123_1C = LinearInterpolation(df_a123_1C.Ahr, df_a123_1C.V, extrapolation_bc=Line())

csv_a123_6C = CSV.File("A123Systems/ANR26650M1B, 6C.csv")
df_a123_6C = DataFrame(csv_a123_6C)
itp_a123_6C = LinearInterpolation(df_a123_6C.Ahr, df_a123_6C.V, extrapolation_bc=Line())

csv_a123_20C = CSV.File("A123Systems/ANR26650M1B, 20C.csv")
df_a123_20C = DataFrame(csv_a123_20C)
itp_a123_20C = LinearInterpolation(df_a123_20C.Ahr, df_a123_20C.V, extrapolation_bc=Line())

## IMPORTANT TO UPDATE WHENEVER THE BATTERIES DATASHEETS ARE CHANGED
## The order is important, it must be done in sequential order by growing discharge rate
discharge_curves_a123 = (
  (1.0,  itp_a123_1C),
  (6.0,  itp_a123_6C),
  (20.0, itp_a123_20C)
);

# Individual cell capacity, in [Ahr]
ahr_a123_cell = 2.65

# Individual cell mass, in [kg]
mass_a123_cell = .076

# Individual cell max discharge rate
c_a123_max = 50

# Individual cell nominal energy capacity, in [kwhr]
kwhr_a123_cell = .00825

discharge_index_vector_a123 = getfield.(discharge_curves_a123, 1)
```

Out[2151]:

```
(1.0, 6.0, 20.0)
```

In [2152]:

```
## Function definitions

# Finds the indices in A which bracket the argument, x. This assumes that A is an ordered s
function findNearestBracket(A,x)
  # find the index which is closest to the value
  tmpIdx = argmin(abs.(A .- x))

  # Find the pair of indices which bracket the value
  if tmpIdx == 1
    lowIdx = 1
    highIdx = 2
  elseif tmpIdx == length(A)
    lowIdx = length(A)-1
    highIdx = length(A)
  else
    if A[tmpIdx] < x
      lowIdx = tmpIdx
      highIdx = tmpIdx+1
    else
      lowIdx = tmpIdx-1
      highIdx = tmpIdx
    end
  end

  return [lowIdx, highIdx]
end

# Calculate by interpolation the instantaneous pack voltage. It takes into account the disc
function instantaneous_pack_voltage(discharge_vector, discharge_curves, c_battery, ahr_accu
  idx = findNearestBracket(discharge_vector, c_battery)
  lowC = discharge_vector[idx[1]]
  highC = discharge_vector[idx[2]]
  v_instantaneous_lowC = discharge_curves[idx[1]][2]
  v_instantaneous_highC = discharge_curves[idx[2]][2]
  v_instantaneous = batt_v(c_battery, lowC, highC, v_instantaneous_lowC(ahr_accum_cell), v_

  return v_instantaneous
end


# Linear interpolation of the battery voltage, with discharge rate as the independent varia
function batt_v(x, x1, x2, y1, y2)
  m = (y2-y1)/(x2-x1)
  y = m*(x-x1) + y1
end
```

Out[2152]:

```
batt_v (generic function with 1 method)
```

User tunable parameters

In [2153]:

```
# pack = "Diamond"
pack = "Molicel P42A"
#pack = "Molicel P26A"

# Number of parallel cells in pack
num_cols_a123 =1

# Number of series rows in pack
num_rows_a123 =19

if pack == "Molicel P42A"
  # Number of parallel cells in pack
  num_cols_lipo = 10
  # Number of series rows in pack
  num_rows_lipo =12

elseif pack == "Molicel P26A"

 # Number of parallel cells in pack
  num_cols_lipo = 10

  # Number of series rows in pack
  num_rows_lipo = 22

elseif pack == "Diamond"
  ahr_diamond_cell = 16  # Can only equal 16, 22, or 30

  # Number of parallel cells in pack
  num_cols_lipo = 1

  # Number of series rows in pack
  num_rows_lipo = 12
end
```

Out[2153]:

```
12
```

In [2154]:

```
if pack == "Molicel P42A"
  discharge_curves_lipo = discharge_curves_P42A;
  mass_lipo_cell = mass_P42A_cell
  ahr_datasheet_lipo_cell = ahr_datasheet_P42A_cell
  ahr_lipo_cell = ahr_datasheet_P42A_cell
  kwhr_lipo_cell = kwhr_P42A_cell
  c_lipo_max = c_P42A_max
elseif pack == "Molicel P26A"
  discharge_curves_lipo = discharge_curves_P26A;
  mass_lipo_cell = mass_P26A_cell
  ahr_datasheet_lipo_cell = ahr_datasheet_P26A_cell
  ahr_lipo_cell = ahr_datasheet_P26A_cell
  kwhr_lipo_cell = kwhr_P26A_cell
  c_lipo_max = c_P26A_max
elseif pack == "Diamond"
  mass_diamond_cell = (1.500/6) * (ahr_diamond_cell/16)  # Backed out from the Foxtech webp

  # Individual cell nominal energy capacity, in [kwhr]
  kwhr_diamond_cell = .111 * (ahr_diamond_cell / ahr_datasheet_diamond_cell)

  discharge_curves_lipo = discharge_curves_diamond;
  mass_lipo_cell = mass_diamond_cell
  ahr_datasheet_lipo_cell = ahr_datasheet_diamond_cell
  ahr_lipo_cell = ahr_datasheet_diamond_cell
  kwhr_lipo_cell = kwhr_diamond_cell
  c_lipo_max = c_diamond_max
end

discharge_index_vector_lipo = getfield.(discharge_curves_lipo, 1)
```

Out[2154]:

```
(0.84, 4.2, 10.0, 20.0, 30.0)
```

In [2155]:

```
# Airplane MTOW
airplaneMass = 300
climbVelocity = 55*.511
glideRatio = 20
```

Out[2155]:

```
20
```

In [2156]:

```
# Diode forward voltage drop
# https://www.vishay.com/docs/93804/vs-175bqg030hf4.pdf
v_drop_diode = 0.782

# Pack nominal energy capacity
kwhr_a123_pack = kwhr_a123_cell * num_cols_a123 * num_rows_a123
kwhr_lipo_pack = kwhr_lipo_cell * num_cols_lipo * num_rows_lipo

# Pack amps parallel capacity
ahr_a123_pack = ahr_a123_cell * num_cols_a123
ahr_lipo_pack = ahr_lipo_cell * num_cols_lipo

# Pack cell mass
mass_a123_pack = mass_a123_cell * num_cols_a123 * num_rows_a123
mass_lipo_pack = mass_lipo_cell * num_cols_lipo * num_rows_lipo


pack_A_label =  string("A123 LiFePo4 (", num_rows_a123, "S", num_cols_a123, "P)")
pack_B_label =  string("Li-ion (", num_rows_lipo, "S, ",num_cols_lipo, "P,", round(ahr_lipo

println("mass [kg]: " * string(round.([mass_a123_pack mass_lipo_pack], digits = 1)))
println("capacity [Ahr / kwHr]: " * string(round.([ahr_a123_pack kwhr_a123_pack;  ahr_lipo_
```

```
mass [kg]: [1.4 8.4]
capacity [Ahr / kwHr]: [2.6 0.2; 42.0 1.8]
```

In [2157]:

```
# Simulate across time
t = 0
delT = .05

# Power schedule
P_TAKEOFF = 18e3
P_MAX_CONTINUOUS = 12e3
P_SUSTAIN_CLIMB = 10e3
P_SUSTAIN = 6e3

# Plotting variables
t_experiment = Float64[]
I_motor = Float64[]
V_a123 = Float64[]
V_lipo = Float64[]
V_bus = Float64[]
soc_a123_experiment = Float64[]
soc_lipo_experiment = Float64[]
c_a123_experiment = Float64[]
c_lipo_experiment = Float64[]
P_experiment = Float64[]
P_a123 = Float64[]
P_lipo = Float64[]
E_a123 = Float64[]
E_lipo = Float64[]
soc_20_lipo = 0
soc_5_a123 = 0
vel_airplane = []
hgt_airplane = []
hgt_dot_airplane = []

c_a123 = 0
c_lipo = 0


## Initial conditions
# Electrical power load
P_elec = 0

# Start off at full charge
soc_a123 = 1
soc_lipo = 1

v_0_a123 = itp_a123_20C(0)
v_0_lipo = discharge_curves_lipo[1][2](0)

kwhr_accum_a123 = 0
kwhr_accum_lipo = 0

ahr_accum_a123 = 0
ahr_accum_lipo = 0

ahr_accum_cell_a123 = ahr_accum_a123/num_cols_a123
ahr_accum_cell_lipo = ahr_accum_lipo/num_cols_lipo

kineticEnergy = 0
height = 0
velocity = sqrt(2 * kineticEnergy / airplaneMass)
```

```
v_instantaneous_a123 = v_0_a123
v_instantaneous_lipo = v_0_lipo

voltage_a123_pack = v_instantaneous_a123 * num_rows_a123
voltage_lipo_pack = (v_instantaneous_lipo * num_rows_lipo) - v_drop_diode

if abs(voltage_lipo_pack - voltage_a123_pack) <= .001
  V_system = voltage_a123_pack
else
  V_system = max(voltage_a123_pack, voltage_lipo_pack )
end

I_a123 = 0
I_lipo = 0

# Low-pass variable for changing the power load
alpha = 0.05
P_setpoint = 0

# Run the simulation
for i=1:(2000/delT)
  t = t + delT

  if 1==0
    # Reduce power according to battery limitations
    if c_lipo > c_lipo_max
      P_elec = P_elec - 1000
    end
  else
    # Reduce power according to schedule and/or battery limitations
    if (t<0.1)
      P_setpoint = 0
    elseif (t<=60)
      P_setpoint = P_TAKEOFF
    elseif (t > 60) && P_setpoint == P_TAKEOFF
      P_setpoint = P_MAX_CONTINUOUS
    elseif (t > 180)  && P_setpoint == P_MAX_CONTINUOUS
      P_setpoint = P_SUSTAIN_CLIMB
    elseif (t > 300)  && P_setpoint == P_SUSTAIN_CLIMB
      P_setpoint = P_SUSTAIN
    end
    P_elec = P_elec + alpha*(P_setpoint-P_elec)
  end


  # Loop enough times to be sure to converge. Convergence is pretty decent after 15 loops,
  if P_elec == 0
    I_a123 = 0;
    I_lipo = 0;

    c_a123 = 0;
    c_lipo = 0;

    v_instantaneous_a123 = instantaneous_pack_voltage(discharge_index_vector_a123, discharg
    v_instantaneous_lipo = instantaneous_pack_voltage(discharge_index_vector_a123, discharg

      # Saturate battery voltages
      if v_instantaneous_lipo > 4.2
        v_instantaneous_lipo = 4.2
      end
```

```
    if v_instantaneous_a123 > 3.4
      v_instantaneous_a123 = 3.4
    end

  voltage_a123_pack = v_instantaneous_a123 * num_rows_a123
  voltage_lipo_pack = (v_instantaneous_lipo * num_rows_lipo) - v_drop_diode

  P_split_pct = 0.5;
else
  # Initialize some variables before going into the convergence loop
  loopCount = 1
  P_split_pct = 0
  seekScalar = .5
  lastRoundLipoHigh = true

  voltage_a123_pack = v_0_a123 * num_rows_a123
  voltage_lipo_pack = (v_0_lipo * num_rows_lipo) - v_drop_diode

  while (loopCount < 100 && abs(v_instantaneous_a123-v_instantaneous_lipo) >= 0.001 )
    I_a123 = (P_elec * (1-P_split_pct)) / voltage_a123_pack
    I_lipo = (P_elec * P_split_pct) / voltage_lipo_pack

    # Calculate cell discharge rate
    c_a123 = I_a123 / ahr_a123_pack
    c_lipo = I_lipo / ahr_lipo_pack

    ## Determine A123 cell voltages
    v_instantaneous_a123 = instantaneous_pack_voltage(discharge_index_vector_a123, discha

    ## Determine LiPo cell voltages
    v_instantaneous_lipo = instantaneous_pack_voltage(discharge_index_vector_lipo, discha

    # Saturate battery voltages
    if v_instantaneous_lipo > 4.2
      v_instantaneous_lipo = 4.2
    end

    if v_instantaneous_a123 > 3.4
      v_instantaneous_a123 = 3.4
    end

    # Determine pack voltages. This is just the number of batteries in series times the v
    voltage_a123_pack = v_instantaneous_a123 * num_rows_a123
    voltage_lipo_pack = (v_instantaneous_lipo * num_rows_lipo) - v_drop_diode

    # Nonlinear search pattern. We're looking for the percentage power split which brings
    if voltage_lipo_pack > voltage_a123_pack
      if (lastRoundLipoHigh == false)
        seekScalar = seekScalar/2
      end

      P_split_pct = P_split_pct + seekScalar

      lastRoundLipoHigh = true
    else
      if (lastRoundLipoHigh == true)
        seekScalar = seekScalar/2
      end

      P_split_pct = P_split_pct - seekScalar
```

```
      lastRoundLipoHigh = false
    end

    if P_split_pct > 1
      P_split_pct =1
    elseif P_split_pct < 0
      P_split_pct = 0
    end

    # Increment loop variables.
    loopCount = loopCount + 1
  end

  # Check if voltages successfully converged. Do this check only when there is power bein
  if abs(voltage_lipo_pack - voltage_a123_pack) > 0.1 && c_lipo > 0.1 && c_a123 > 0.1
    println("Voltages did not converge at t=" * string(round(t, digits = 2)) * ", voltage
     break;
  end
end

# Update the system voltage
if abs(voltage_lipo_pack - voltage_a123_pack) <= .001
  V_system = voltage_a123_pack
else
  # This is the case where the A123Systems pack hasn't discharged down to the LiPo/Li-ion
  V_system = max(voltage_a123_pack, voltage_lipo_pack )
end

# Calculate accumulated amp-hour consumption
ahr_accum_a123 = ahr_accum_a123 + I_a123 * delT/3600
ahr_accum_lipo = ahr_accum_lipo + I_lipo * delT/3600

# Calculate accumulated discharge from individual cells
ahr_accum_cell_a123 = ahr_accum_a123/num_cols_a123
ahr_accum_cell_lipo = ahr_accum_lipo/num_cols_lipo

# Calculate instantaneous power
P_instantaneous_a123 = I_a123 * V_system
P_instantaneous_lipo = I_lipo * V_system

# Calculate accumulated kw-hour consumption
kwhr_accum_a123 = kwhr_accum_a123 + P_instantaneous_a123 * delT/3600/1000
kwhr_accum_lipo = kwhr_accum_lipo + P_instantaneous_lipo * delT/3600/1000

# Calculate State of Charge
soc_a123 = 1 - ahr_accum_a123/ahr_a123_pack
soc_lipo = 1 - ahr_accum_lipo/ahr_lipo_pack

# Stop the experiment if a pack is dead
if (ahr_accum_a123 > ahr_a123_pack && (num_cols_a123 * num_rows_a123 > 0)) || ahr_accum_l
    println("[BATTERY EMPTY] t: " * string(round(t, digits = 1)) * ", A123: " * string(ro
    break
end

# Mark the point in time when the A123Systems pack's SoC goes below 5%
if soc_5_a123 == 0 && soc_a123 < 0.05
  soc_5_a123 = t
end

# Mark the point in time when the LiPo/Li-ion pack's SoC goes below 20%
```

```
if soc_20_lipo == 0 && soc_lipo < 0.20
  soc_20_lipo = t
end

# End the simulation one minute after the LiPo/Li-ion
if soc_20_lipo !=0 && (t-soc_20_lipo) > 60
  break
end



# Simulate airplane. This is an extremely simplified model, but it works fine for these p
drag = airplaneMass * 9.805/glideRatio * (velocity/climbVelocity)^2
if (velocity < climbVelocity)
  # Below climbVelocity, we haevn't taken off yet
  eff = .4
  kineticEnergy = kineticEnergy + (P_elec * eff - velocity * drag) * delT
  velocity = sqrt(2 * kineticEnergy / airplaneMass)

  h_dot = 0
else
  # Above climbVelocity, we're putting excess energy into climbing
  eff = .8*.90  # Propller efficiency times electrical efficiency
  h_dot = (P_elec * eff - velocity * drag)/(airplaneMass*9.805)
  height = height + h_dot*delT
end

# Some useful console spew
if 1==0
  println(
      t, ": ",
      " [",
      round(v_instantaneous_a123, digits=3), " ",
      round(voltage_a123_pack, digits=2), " ",
      round(c_a123, digits=1), ", ",
      round(ahr_accum_a123, digits=4), ", ",
      round(soc_a123*100, digits=1),
      "], [",
      round(v_instantaneous_lipo, digits=3), " ",
      round(voltage_lipo_pack, digits=2), ", ",
      round(c_lipo, digits=1), ", ",
      round(ahr_accum_lipo, digits=4), ", ",
      round(soc_lipo*100, digits=1),
  " ], :", round(voltage_a123_pack*I_a123 + voltage_lipo_pack*I_lipo))
end


append!(t_experiment, t)
append!(c_a123_experiment, c_a123)
append!(c_lipo_experiment, c_lipo)
append!(soc_a123_experiment, soc_a123)
append!(soc_lipo_experiment, soc_lipo)
append!(V_a123, voltage_a123_pack)
append!(V_lipo, voltage_lipo_pack)
append!(V_bus, V_system)
append!(P_experiment, P_elec)
append!(P_a123, P_instantaneous_a123)
append!(P_lipo, P_instantaneous_lipo)
append!(E_a123, kwhr_accum_a123)
append!(E_lipo, kwhr_accum_lipo)
append!(vel_airplane, velocity)
```

```
  append!(hgt_airplane, height)
  append!(hgt_dot_airplane, h_dot)

end

[soc_5_a123, soc_20_lipo]
```

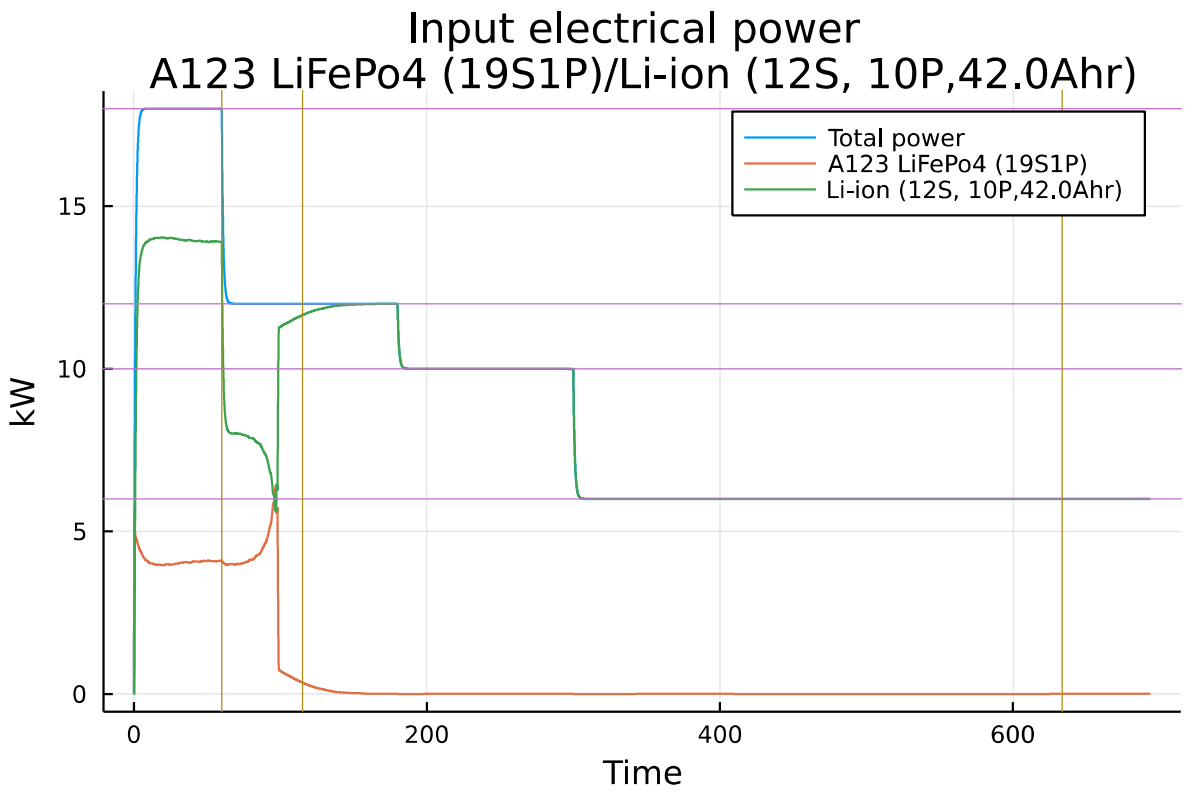
Out[2157]:

```
2-element Vector{Float64}:
 115.2499999999956
 633.7499999999714
```

In [2158]:

```
plot(t_experiment, P_experiment./1000, xlabel = "Time", ylabel = "kW", ylims=[0,Inf], title

plot!(t_experiment, [P_a123 P_lipo]./1000, label=[pack_A_label pack_B_label])

hline!([P_TAKEOFF, P_MAX_CONTINUOUS, P_SUSTAIN_CLIMB, P_SUSTAIN]./1000, lw=0.5, label="")
vline!([60, soc_5_a123, soc_20_lipo], lw=0.5, label="")
```

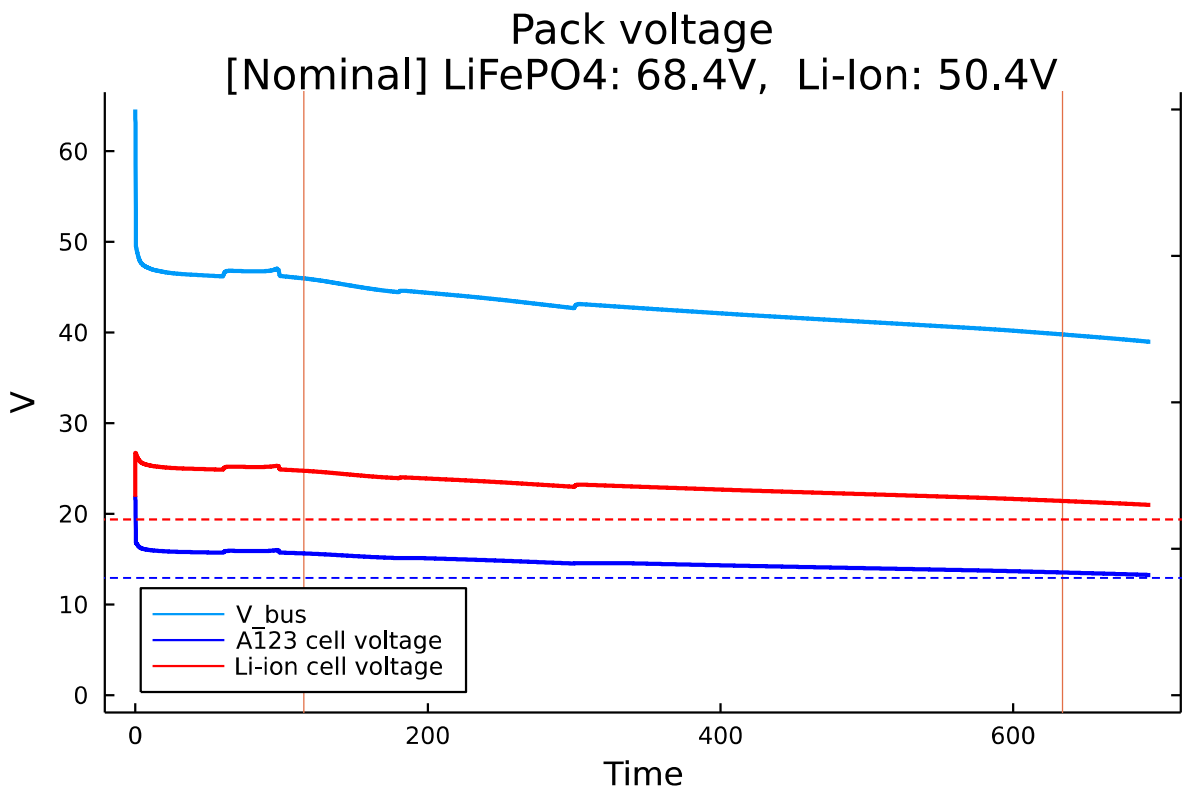
Out[2158]:



In [2159]:

```
colorRightAxis = [:blue :red]
linestyleRightAxis = :solid



plot(t_experiment, [V_bus], xlabel = "Time", title=("Pack voltage\n[Nominal] LiFePO4: " * s

# Plot the times when the packs are empty
vline!([soc_5_a123, soc_20_lipo], lw=0.5, label="")

# Ugly hack to get all the graph labels on the same legend
plot!(1, [NaN NaN], label=["A123 cell voltage" "Li-ion cell voltage"], grid=false, linestyl

p = twinx()

plot!(p, t_experiment, [V_a123./num_rows_a123 V_lipo./num_rows_lipo], legend=false, xticks=

hline!(p, [2.0 3.0], label="", linestyle=:dash, color=[:blue :red])  # Plot minimum cell vo
```

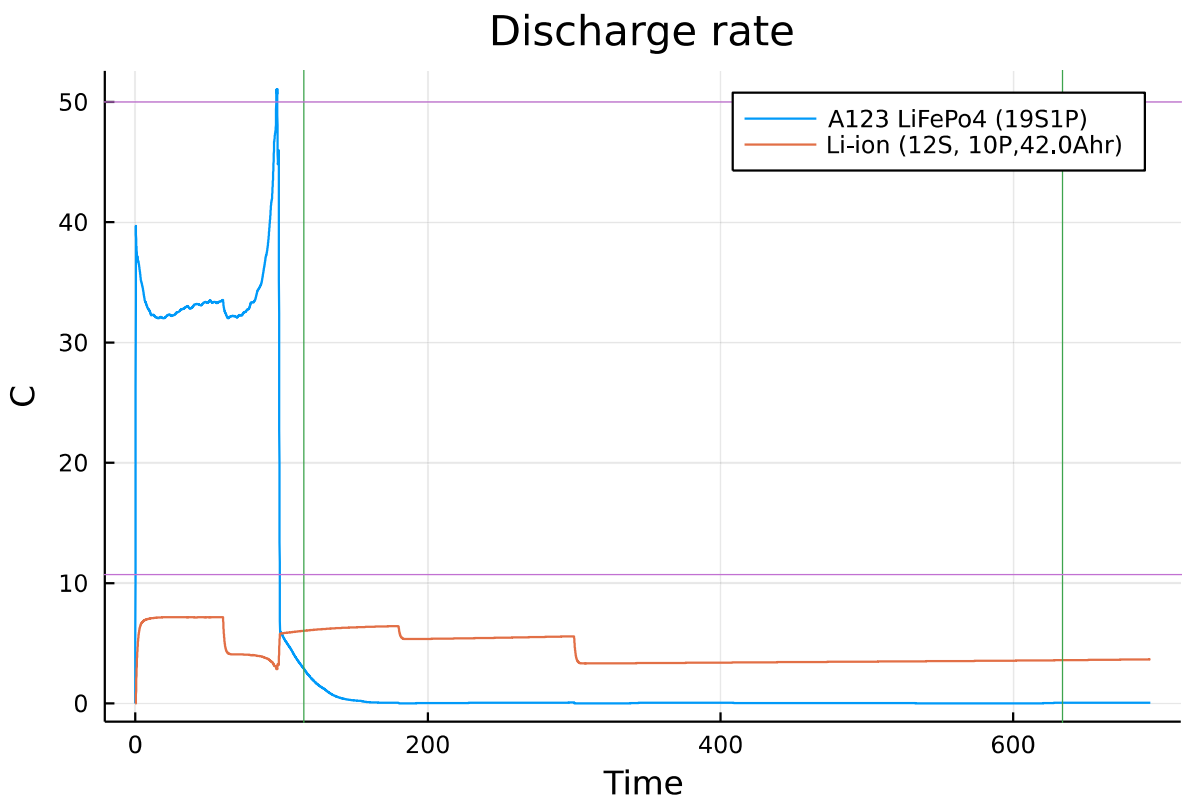
Out[2159]:



In [2160]:

```
plot(t_experiment, [c_a123_experiment c_lipo_experiment], title="Discharge rate", xlabel =

vline!([soc_5_a123, soc_20_lipo], lw=0.5, label="")
hline!([c_a123_max, c_lipo_max], label="", lw=0.5) # Max nominal discharge rates
# savefig("plot.png")
```

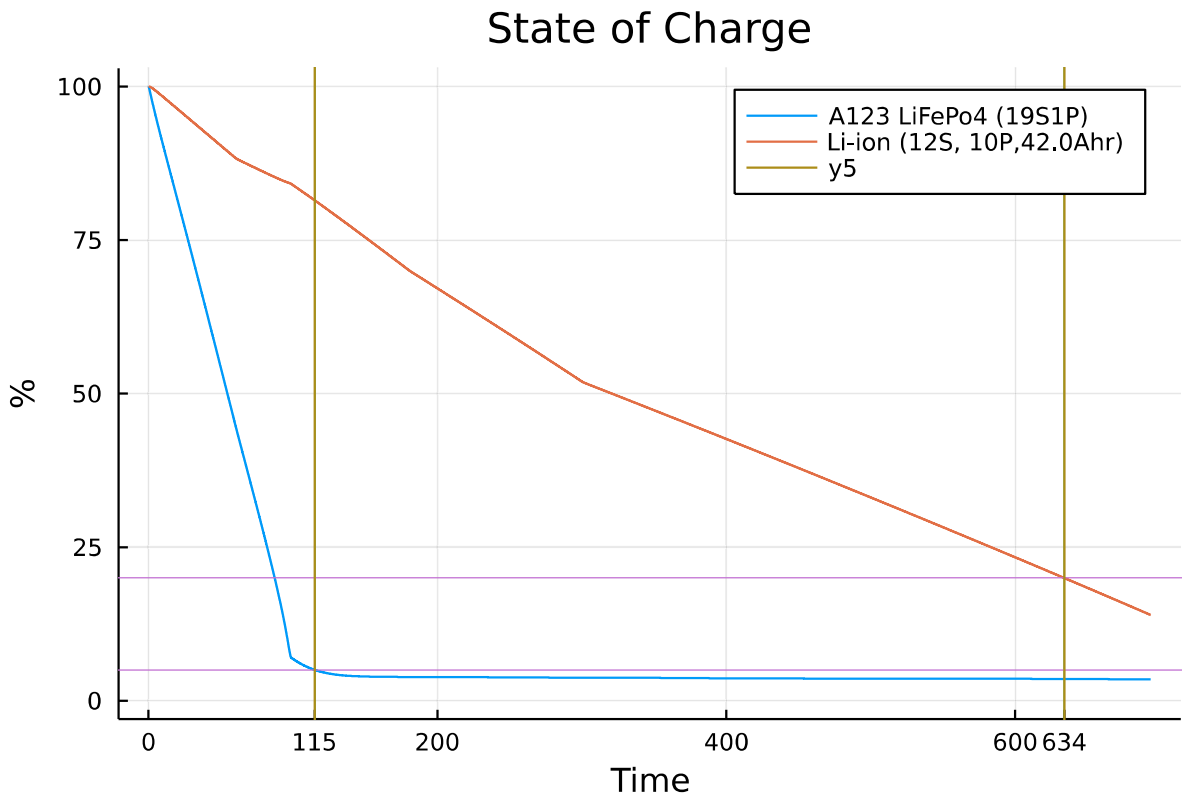
Out[2160]:



In [2161]:

```
b = plot(t_experiment, [soc_a123_experiment soc_lipo_experiment].*100, title="State of Char

vline!([soc_5_a123, soc_20_lipo], lw=0.5, label="")
hline!([5,20], label="", lw=0.5)  # SoC limits

old_xticks = xticks(b[1]) # grab xticks of the 1st subplot
new_xticks = (round.([soc_5_a123, soc_20_lipo]), string.(Int.(round.([soc_5_a123, soc_20_li
vline!(new_xticks[1])
keep_indices = findall(x -> all(x .≠ new_xticks[1]), old_xticks[1])
merged_xticks = (old_xticks[1][keep_indices] ∪ new_xticks[1], old_xticks[2][keep_indices] ∪
xticks!(merged_xticks)
```

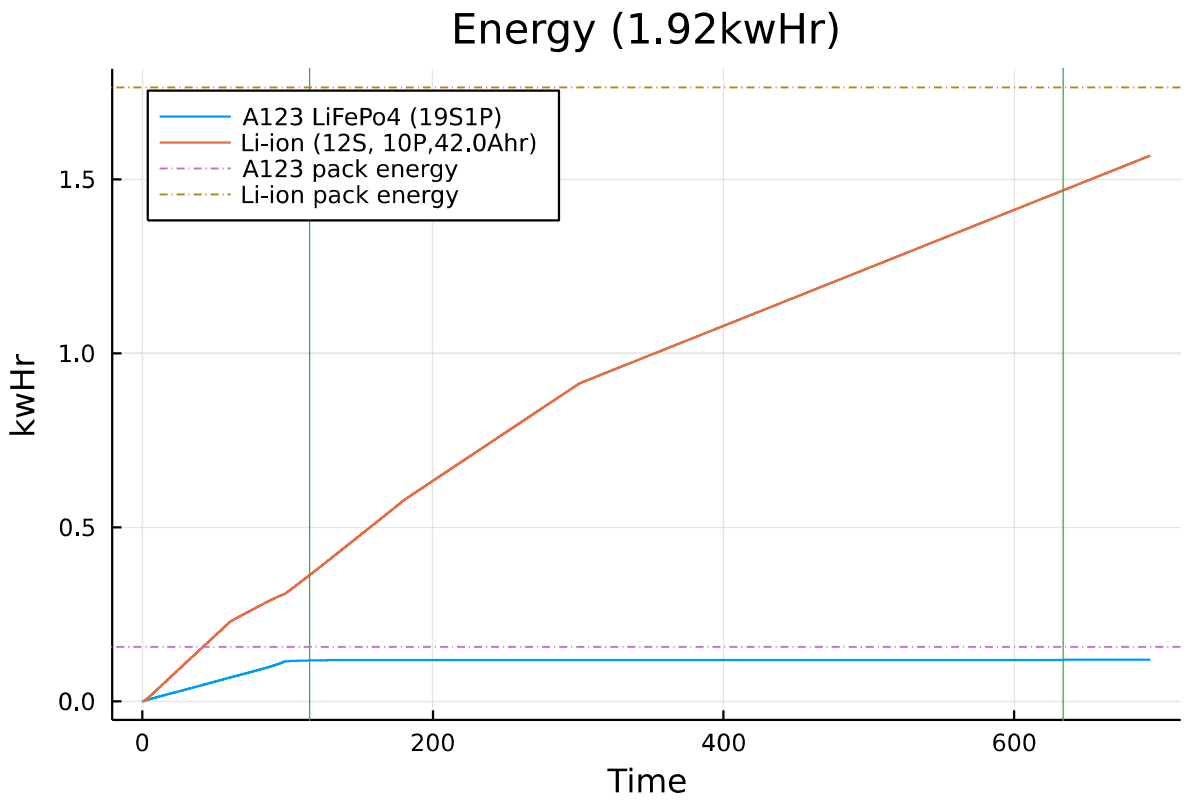
Out[2161]:



In [2162]:

```
title = "Energy (" * string(round(kwhr_a123_pack + kwhr_lipo_pack, digits = 2)) * "kwHr)"
plot(t_experiment, [E_a123 E_lipo], title=title, xlabel = "Time", ylabel = "kwHr", label=[p

vline!([soc_5_a123, soc_20_lipo], lw=0.5, label="")
hline!([kwhr_a123_pack], label="A123 pack energy", linestyle=:dashdot)
hline!([kwhr_lipo_pack], label="Li-ion pack energy", linestyle=:dashdot)
```

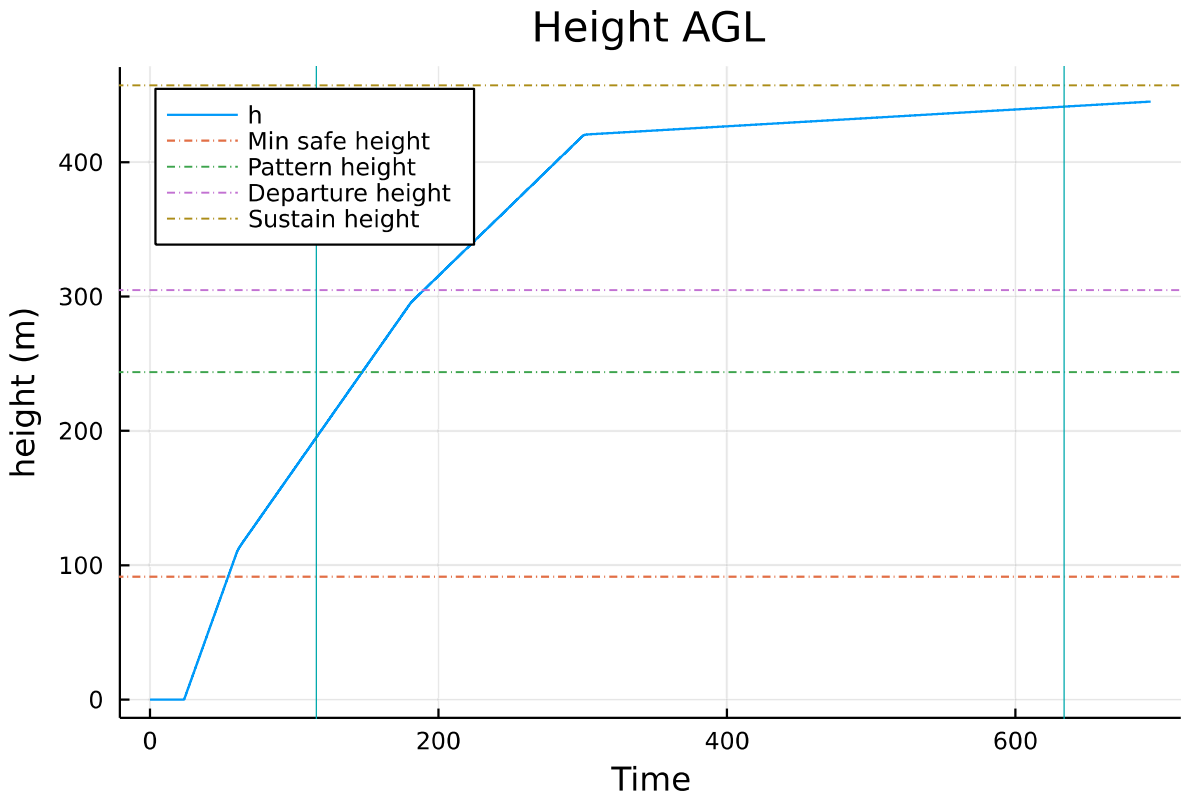
Out[2162]:



In [2163]:

```
plot(t_experiment, hgt_airplane, xlabel = "Time", ylabel = "height (m)", xlims=[0,Inf], tit

hline!([300*12*.0254], label="Min safe height", linestyle=:dashdot)
hline!([800*12*.0254], label="Pattern height", linestyle=:dashdot)
hline!([1000*12*.0254], label="Departure height", linestyle=:dashdot)
hline!([1500*12*.0254], label="Sustain height", linestyle=:dashdot)

vline!([soc_5_a123, soc_20_lipo], lw=0.5, label="")
```

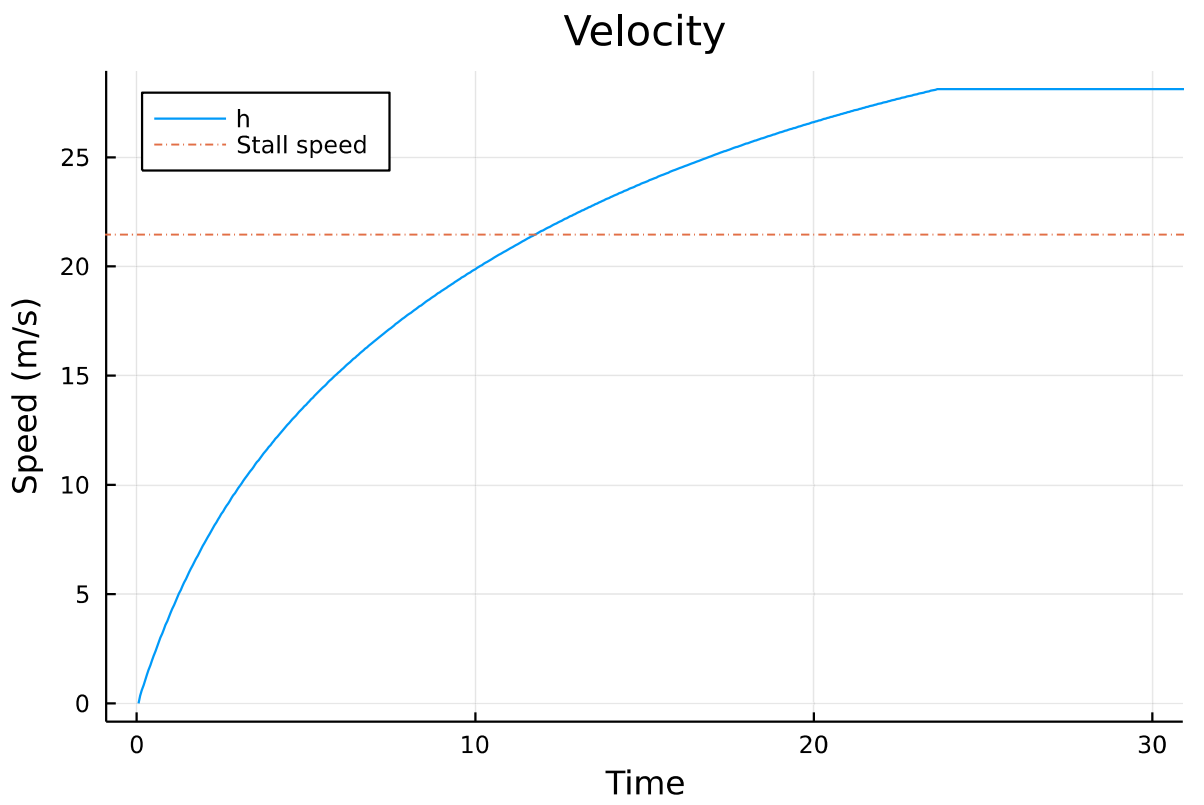
Out[2163]:



In [2164]:

```
plot(t_experiment, vel_airplane, xlabel = "Time", xlims=[0,30], ylabel = "Speed (m/s)", yli

hline!([42*.511], label="Stall speed", linestyle=:dashdot)
```

Out[2164]:



In [ ]:

In [ ]:

In [ ]: